Приложения Пролога. Экспертные системы

Понятие экспертной системы

Экспертная система (ЭС) – компьютерная система, использующая знания эксперта для высокоэффективного решения задач в проблемной области, для которой традиционные формальные методы решения неизвестны или неприменимы вследствие имеющихся ограничений.

Первые ЭС начали разрабатываться в середине 60-х годов прошлого века для решения задач медицинской диагностики (система MYCIN), определения структуры сложных молекул по данным масс-спектрограмм (система DENDRAL), определения залежей полезных ископаемых (система PROSPECTOR) и др. В течение 70-х и 80-х годов прошлого века шло активное развитие и формирование инженерии знаний, как важнейшего направления в рамках искусственного интеллекта (ИИ). В настоящее время ЭС широко используется в самых различных областях.

Отличительной чертой данного класса систем является использование для решения задач знаний опытного эксперта.

Кассы задач, в которых используются ЭС:

- *интерпретация* – составление смыслового описания ситуации по наблюдаемым данным – распознавание образов, понимание речи и т. п. (SPE - определение концентрации гамма-глобулина в крови);
- медицинская и техническая *диагностика* – определение причин неисправностей по результатам наблюдений (MYCIN - диагностика бактериальных инфекций);
- *прогнозирование* – определение вероятных последствий наблюдаемых ситуаций – предсказание погоды, урожая, курса валют и т.п. (PLANT/cd - определения потерь урожая от черной совки);
- *планирование* – определение последовательности действий, приводящих к желаемой цели – планирование действий робота, маршрута движения (TATR - планирование авиаударов по аэродромам противника);
- *управление* – целенаправленное воздействие на объект (применяется в задачах, где традиционные модели автоматического управления неприменимы или неэффективны: управление деловой активностью, боем, воздушным движением и т.п.);

- *мониторинг* – сравнение результатов наблюдений с ожидаемыми или желаемыми (медицинский и экологический мониторинг, атомные электростанции);
- *обучение* – диагностика, формирование и коррекция знания и навыков обучаемого GUIDON - обучение студентов-медиков (антибактериальная терапия);.
- о*тладка* - составление рецептов исправления неправильного функционирования системы. ONCOCIN - планирование химиотерапевтического лечения;
- р*емонт* - выполнение последовательности предписанных исправлений. TQMSTUNE - настройка масс-спектрометра.
- *проектирование* - построение конфигурации объектов при заданных ограничениях. XCON (R1) - выбор оптимальной конфигурации аппаратных средств (VAX).

На рис. 1 представлена обобщенная архитектура ЭС.

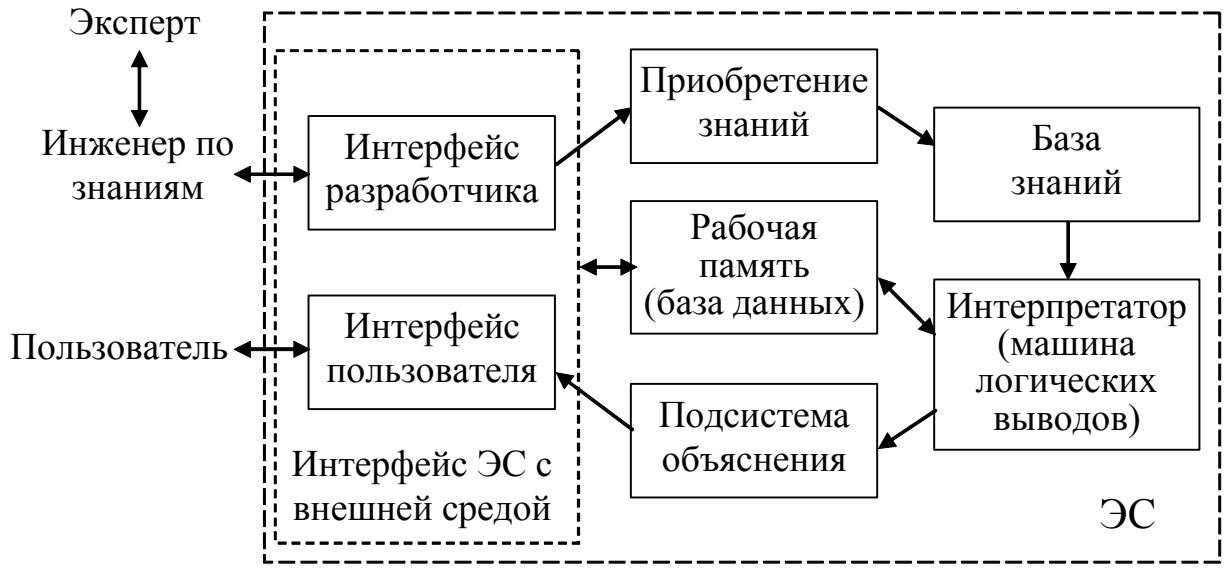

Рис. 1. Обобщенная архитектура экспертной системы

Интерфейс ЭС с внешней средой поддерживает взаимодействие ЭС с внешним миром на всех стадиях жизненного цикла системы и включает две компоненты: *интерфейс разработчика* и *интерфейс пользователя*. Интерфейс разработчика используется на этапе разработки ЭС, его основной функцией является поддержка процесса наполнения базы знаний (БЗ). Обычно эта функция выполняется экспертом в предметной области во взаимодействии с инженером по знаниям. Интерфейс пользователя поддерживает общение

пользователя с системой в режиме консультации или взаимодействие ЭС с техническими средствами (в случае ее встроенного применения) на этапе ее использования.

Компонента приобретения знаний предназначена для занесения в БЗ новых знаний и модификации имеющихся, как на этапе начального обучения ЭС, так и в режиме ее дообучения в процессе эксплуатации. Ее задачей, в частности, является преобразование знаний в форму, позволяющую машине логических выводов (МЛВ) использовать их в процессе работы.

Рабочая память или *база данных* (БД) хранит факты о текущем состоянии предметной области, промежуточных и окончательных результатах вывода.

База знаний служит для хранения знаний о проблемной области. Форма хранения соответствует выбранной модели представления знаний.

Машина логических выводов (МЛВ) или *интерпретатор* осуществляет вывод решения задачи на основе имеющихся в системе знаний и фактов. БД, БЗ и МЛВ составляют ядро ЭС.

Подсистема объяснения обеспечивает трассировку хода вывода решения и предоставление по требованию пользователя объяснения вывода с нужной степенью детализации. Эта функция исключительно важна для ЭС, т.к. при принятии ответственных решений на основе рекомендаций ЭС пользователь, как правило, желает знать, каким образом они были получены.

Этапы разработки и стадии жизненного цикла ЭС

В процессе разработки ЭС принято выделять пять взаимодействующих и частично пересекающихся этапов: идентификация, концептуализация, формализация, реализация и тестирование.

Этап *идентификации* – происходит осмысление необходимости решения задачи методами инженерии знаний, уточняются цели и задачи ЭС, определяются участники процесса разработки и их роли, а также требуемые ресурсы, в том числе возможные источники знаний.

Этап *концептуализации* – строится концептуальная модель проблемной области, т. е. выделяются ключевые понятия, свойства и отношения, необходимые для описания процесса решения задачи. Задача инженера по знаниям на этом этапе состоит в том, чтобы определить, достаточно ли выделенных ключевых понятий и отношений для описания всех имеющихся примеров.

Этап *формализации* – построенная концептуальная модель представляется с использованием выбранных формальных моделей представления знаний. На этом же этапе принимается решение о выборе инструментальных программных средств проектирования ЭС, либо о разработке своих собственных.

Этап *реализации* – создается один или несколько прототипов ЭС, решающих требуемые задачи. На этом этапе выбираются структуры данных и реализуются правила вывода и управляющие стратегии, принятые на этапе формализации, устраняются несоответствия между спецификациями структур данных, правил и схем управления.

Этап *тестирования* – осуществляется оценка работы программы-прототипа на различных входных воздействиях, выявляются ситуации неадекватных решений и определяются их причины.

По степени проработанности и отлаженности в жизненном цикле ЭС выделяют пять стадий: демонстрационный прототип, исследовательский прототип, действующий прототип, промышленная система и коммерческая система.

Демонстрационный прототип решает часть требуемых задач, подтверждая потенциальную применимость методов инженерии знаний и принципиальную осуществимость разработки.

Исследовательский прототип решает все требуемые задачи, но не полностью отлажен и неустойчив в работе.

Действующий прототип надежно решает все задачи, но для решение сложных задач может потребоваться чрезмерно много времени и/или памяти.

Промышленная система обеспечивает высокое качество, надежность, быстродействие и эффективность работы в реальных условиях эксплуатации.

Коммерческая система пригодна для продажи различным потребителям.

Две стратегии механизма логического вывода на множестве правил БЗ ЭС

Существуют два основных подхода к организации процесса логического вывода на множестве правил базы знаний экспертной системы.

Первый подход в качестве основы использует поставленную цель и осуществляет поиск в базе знаний подтверждающих или опровергающих данную цель фактов посредством имеющихся в базе правил. Такой подход называется ***обратной стратегией*** логического вывода, или стратегией "от цели к фактам".

Рассмотрим пример реализации на языке Visual Prolog фрагмента экспертной системы с обратной стратегией вывода. Рассматривается задача постановки медицинского диагноза по имеющимся симптомам и показателям самочувствия человека. Вот пример диалога, который происходит при работе программы:

```
Имеется высокая температура?
Нет
Чувствуются боли в горле?
Да
Наблюдается слюнотечение?
Да
Ваш диагноз – инородное тело в гортани.
```

В соответствии с синтаксисом языка Visual Prolog раздел `DOMAINS` объявляет типы данных для программы. Раздел `PREDICATES` объявляет предикаты, тела которых даются в разделе `CLAUSES`. Этот последний раздел составляет исполняемую программу. В разделе `GOAL` записывается предикат запроса к программе. Факты и правила завершаются символом "." (точка). Переменные начинаются с заглавной буквы.

Введена группа правил с предикатом `diagnosis` в голове правила, каждое из которых описывает один диагноз (строки 27–58). Данные правила построены с использованием правил более низкого уровня с предикатом `it_is`, описывающих группы симптомов, общих для нескольких диагнозов (строки 59–66). Иерархия таких правил может быть продолжена.

Предикат `positive` в правых частях этих правил проверяет наличие или отсутствие необходимой для диагноза информации в базе данных (строки 17, 18). В случае, если в базе данных отсутствует информация о симптоме (утвердительная или отрицательная), пользователю задается вопрос (строка 18). Для этой цели в программе присутствует предикат `ask` (строки 19, 20). Ответ "да" или "нет" с помощью предиката `remember` и стандартного предиката `asserta` (строки 21, 22) сохраняется в базе данных, структура записей в которой описана в разделе `DATABASE` (строки 1–3).

Программа начинает работу с проверки главной цели, описываемой предикатом `run` (строки 67, 68 и 13–16).

В случае невозможности доказательства цели `diagnosis` работает второе правило для `run` (строки 15, 16).

Последними в списке предикатов в правых частях правил для предиката run располагаются предикаты `clear_facts`, правила для которых с помощью стандартного предиката `retractall` удаляют из базы данных все факты, добавленные в течение текущего сеанса экспертизы (строки 23, 24).

Это делается для того, чтобы при реализации следующих запросов из базы данных не использовались факты, накопленные в процессе реализации предыдущих запросов. Последнее из правил для предиката `clear_facts` выводит подсказку и ждет от пользователя нажатия Enter (строки 25, 26). После этого выводится фраза `yes`, которая является признаком достижения главной цели `run`.

```
1) DATABASE
2) xpositive(string,string)
3) xnegative(string,string)
4) PREDICATES
5) ask(string,string)
6) nondeterm diagnosis(string)
7) it_is(string)
8) positive(string,string)
9) run
10) clear_facts
11) remember(string,string,string)
12) CLAUSES
13) run :- diagnosis(X),!,nl,write("Ваш диагноз - "),
14)        write(X),write("."),nl,nl,clear_facts.
15) run :- nl,write("Система не может поставить
16) диагноз."),nl,clear_facts.
17) positive(X,Y) :- xpositive(X,Y),!.
18) positive(X,Y) :- not(xnegative(X,Y)), ask(X,Y).
19) ask(X,Y) :- write(X), write(" "), write(Y),
20) write("?"),nl,readln(Reply),remember(X,Y,Reply).
21) remember(X,Y,"да") :- asserta(xpositive(X,Y)).
22) remember(X,Y,"нет") :-asserta(xnegative(X,Y)),fail.
23) clear_facts :- retractall(xpositive(_,_)),fail.
24) clear_facts :- retractall(xnegative(_,_)),fail.
```

```
25) clear_facts :- write("Нажмите Enter для выхода."),
26) readln(_).
27) diagnosis("Ангина") :- it_is("Предположение2"),
28) positive("Имеется","налет на миндалинах").
29) diagnosis("Инородное тело в гортани") :-
30) positive("Чувствуются","боли в горле"),
31) positive("Наблюдается","слюнотечение"),
32) not(positive("Имеется","высокая температура")).
33) diagnosis("Острый ларингит") :-
34) positive("Наблюдается","кашель"),
35) positive("Наблюдается","затрудненное дыхание"),
36) not(positive("Имеется","налет на миндалинах")).
37) diagnosis("Коклюш") :- it_is("Предположение3"),
38) positive("Имеется","высокая температура"),
39) positive("Происходит","выделение мокроты при кашле").
40) diagnosis("Пищевая интоксикация") :-
41) it_is("Предположение1"),
42) positive("Чувствуется","тошнота"),
43) positive("Наблюдается", "плохой аппетит").
44) diagnosis("Скарлатина") :- it_is("Предположение2"),
45) positive("Имеется","сыпь на коже").
46) diagnosis("Острый насморк") :-
47) it_is("Предположение1"),
48) positive("Чувствуется","жжение в носу").
49) diagnosis("Отит") :- it_is("Предположение1"),
50) positive("Чувствуются","боли в ухе"),
51) positive("Наблюдаются","симптомы интоксикации").
52) diagnosis("ОРЗ") :- it_is("Предположение3"),
53) positive("Чувствуется","головная боль").
54) diagnosis("Серная пробка в ухе") :-
55) positive("Чувствуется","тошнота"),
56) positive("Чувствуется","понижение слуха при
57) попадании в ухо воды"),
58) positive("Чувствуется","головная боль").
59) it_is("Предположение1") :-
60) positive("Имеется","высокая температура"),
61) positive("Чувствуется","головная боль").
```

```
62) it_is("Предположение3") :-
63) positive("Наблюдатся","насморк"),
64) positive("Наблюдается","кашель").
65) it_is("Предположение2") :- it_is("Предположение1"),
66) positive("Чувствуются","боли в горле").
67) GOAL
68) run.
```

Вторая стратегия берет за основу имеющиеся правила и факты и строит новые факты, которые являются их логическими следствиями. Эти факты добавляются к имеющимся, и процесс повторяется. Процесс заканчивается при достижении факта, совпадающего с поставленной исходной целью, или при невозможности дальнейшего пополнения множества фактов. Этот подход называется *прямой стратегией* логического вывода, или стратегией "от фактов к цели". На практике используется дополнительный механизм выбора правил из набора правил с истинными посылками, от которого зависит, какие из фактов будут добавлены в базу данных. Далее приведена реализация на Прологе решения той же самой задачи постановки медицинского диагноза. Здесь запрограммирована серия проверок симптомов, реализованных предикатами `test1` … `test5`, каждая из которых позволяет отбросить примерно половину возможных решений. Блок-схема такой серии проверок изображена на рис. 2 и 3.

Организация вопросов пользователю и сохранения ответов в базе данных точно повторяет организацию вопросов в версии с обратной стратегией вывода (строки 28–32, 3, 4).

Разница в первом правиле для предиката главной цели `run` по сравнению с обратной стратегией заключается в использовании в правой части этого правила предиката `found`, предназначенного для сохранения в базе данных факта установленного диагноза (см. строки 2 и 19). Правило для предиката `find_diagnosis` (строки 23, 24) кардинально отличается от правила предиката `diagnosis` для обратной стратегии вывода тем, что в его правой части последовательно записаны предикаты правил `test1` … `test5`. Данные правила проверяют симптомы болезни и запрашивают информацию у пользователя в соответствии с уровнями, отображенными на блок-схеме. Таких уровней в данном примере 5, т. е. требуется максимум пять проверок симптомов, чтобы установить диагноз. Аргументы данных предикатов

отражают накопление истории уже выполненных проверок, и их количество равно количеству пройденных уровней плюс 1. Правила для данных предикатов (строки 38–89) отображают все варианты решений на каждом из уровней, а комбинации аргументов отражают историю ответов, накопленную к моменту, когда применяется правило `test1 … test5`.

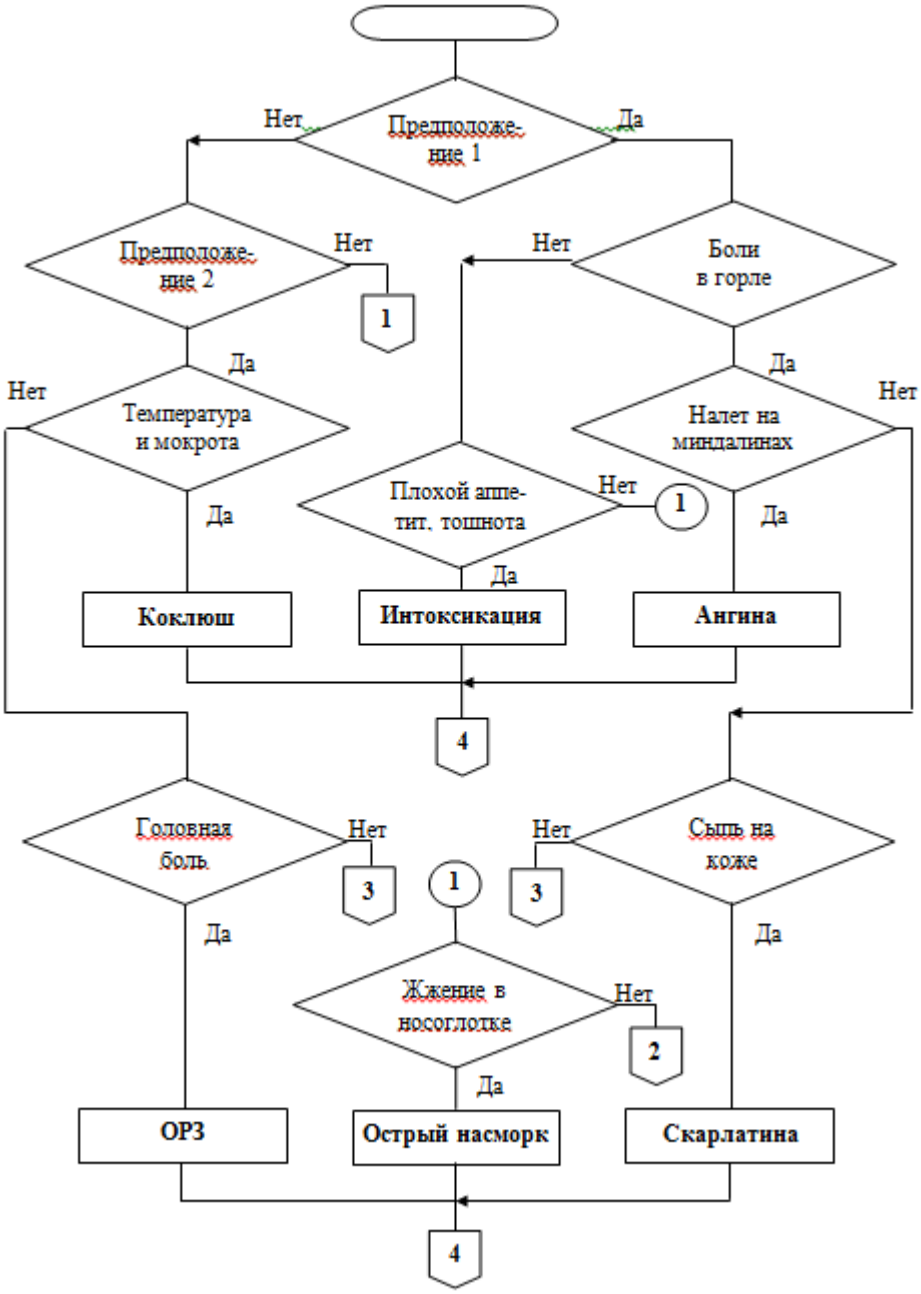

Рисунок 2

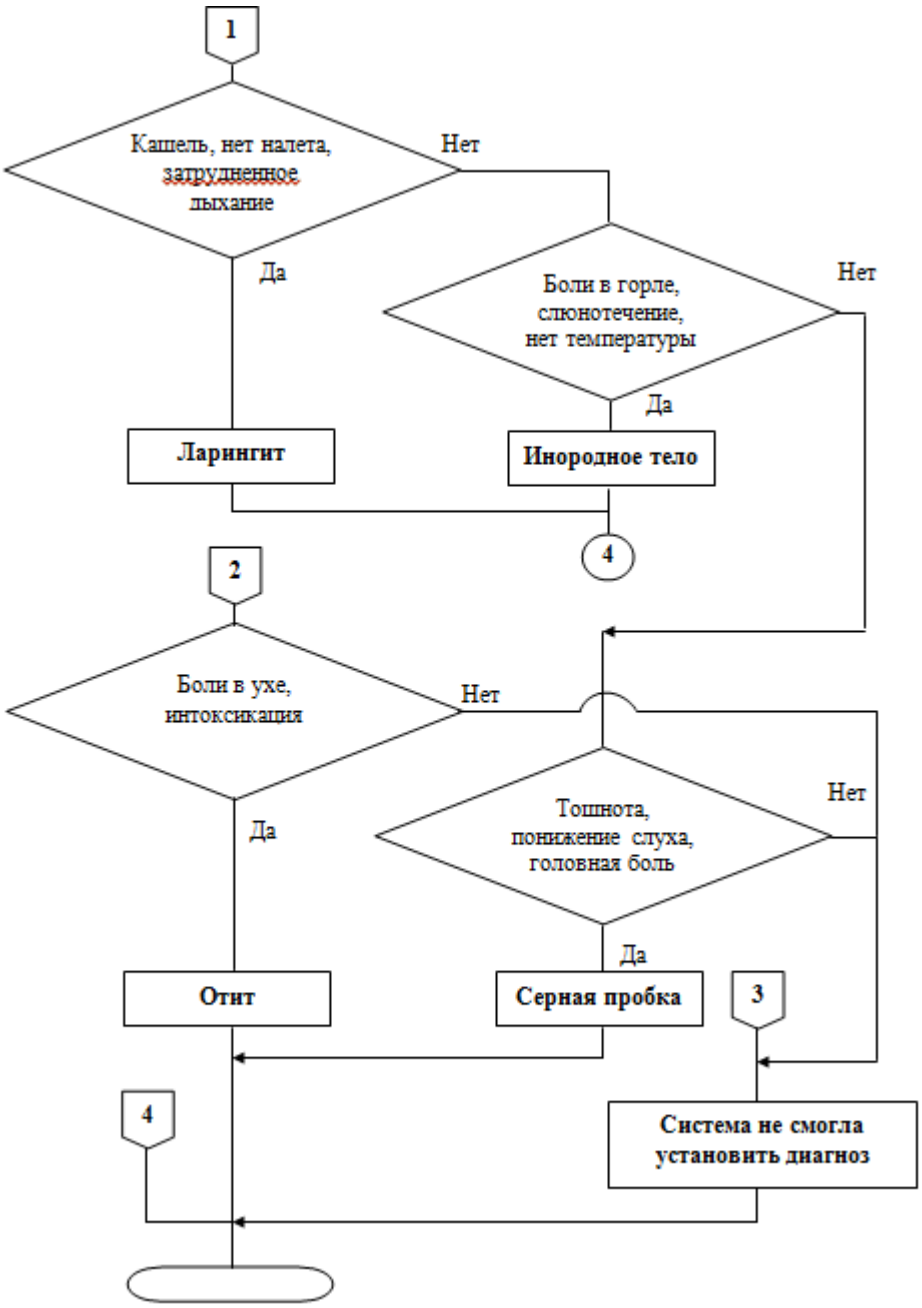

Рисунок 3

Например, правило с заголовком test3("a", "n", "g") (строки 49–52) отражает проверку третьего уровня, до которой от пользователя были получены ответы о наличии следующих симптомов: высокая температура и головная боль (параметр "a" в первой позиции), боль в горле отсутствует (параметр "n" во второй позиции). Это правило выполнится успешно при получении от пользователя ответа о том, что имеются симптомы тошноты и низкого аппетита (эта ситуация кодируется параметром "g"), и последний стандартный предикат asserta в теле этого правила запишет в базу данных установленный диагноз "пищевая интоксикация". В теле правил для предикатов test1 и test2 (строки 38,

43) используются вспомогательные предикаты it_is, которые предназначены для проверки одновременного наличия нескольких симптомов (строки 90–95).

Сохранение в базе данных (строки 3, 4) ответов пользователя о наличии или отсутствии симптомов позволяет не задавать пользователю повторных вопросов в случае, когда один и тот же симптом свойственен нескольким болезням.

```
1) DATABASE
2) found(string)
3) xpositive(string,string)
4) xnegative(string,string)
5) PREDICATES
6) ask(string,string)
7) it_is(string)
8) positive(string,string)
9) run
10) clear_facts
11) remember(string,string,string)
12) find_diagnosis
13) test1(string)
14) test2(string,string)
15) test3(string,string,string)
16) nondeterm test4(string,string,string,string)
17) test5(string,string,string,string,string)
18) CLAUSES
19) run :- find_diagnosis,found(X),
20)       write("Ваш диагноз - ",X),nl,clear_facts,!.
21)  run:-nl,write("Система не может поставить диагноз."),
22) nl, clear_facts.
23) find_diagnosis :- test1(X),test2(X,Y),test3(X,Y,Z),
24)       test4(X,Y,Z,S),test5(X,Y,Z,S,_),!.
25) find_diagnosis.
26) positive(X,Y) :- xpositive(X,Y),!.
27) positive(X,Y) :- not(xnegative(X,Y)), ask(X,Y).
28) ask(X,Y) :- write(X), write(" "), write(Y),
29)             write("?"), nl,readln(Reply),
30)             remember(X,Y,Reply).
```

```
31) remember(X,Y,"да") :- asserta(xpositive(X,Y)).
32) remember(X,Y,"нет") :-asserta(xnegative(X,Y)),fail.
33) clear_facts :- retractall(xpositive(_,_)),fail.
34) clear_facts :- retractall(xnegative(_,_)),fail.
35) clear_facts :- retractall(found(_)),fail.
36) clear_facts :- write("Нажмите Enter для выхода."),
37)                readln(_).
38) test1("a") :- it_is("Предположение1"),!.
39) test1("n").
40) test2("a","b") :- positive("Чувствуются",
41) "боли в горле"),!.
42) test2("a","n").
43) test2("n","c") :- it_is("Предположение2"),!.
44) test2("n","n").
45) test3("a","b","d") :- positive("Имеется",
46) "налет на миндалинах"),
47) asserta(found("ангина")),!.
48) test3("a","b","n").
49) test3("a","n","g") :- positive("Чувствуется",
50) "тошнота"),
51) positive("Наблюдается", "плохой аппетит"),
52) asserta(found("Пищевая интоксикация")),!.
53) test3("a","n","n").
54) test3("n","c","e") :- positive("Имеется",
55) "высокая температура"),
56) positive("Происходит",
57) "выделение мокроты при кашле"),
58) asserta(found("Коклюш")),!.
59) test3("n","c","n").
60) test3("n","n","f"):-positive("Наблюдается","кашель"),
61) positive("Наблюдается","затрудненное дыхание"),
62) not(positive("Имеется","налет на миндалинах")),
63) asserta(found("Острый ларингит")),!.
64) test3("n","n","n").
65) test4("a","b","n","h") :- positive("Имеется",
66) "сыпь на коже"), asserta(found("Скарлатина")),!.
67) test4("a","n","n","i") :-
```

```
68) positive("Чувствуется","жжение в носу"),
69) asserta(found("острый насморк")),!.
70) test4("a","n","n","n").
71) test4("n","c","n","j") :-
72) positive("Чувствуется","головная боль"),
73) asserta(found("ОРЗ")),!.
74) test4("n","n","n","k") :-
75) positive("Чувствуются","боли в горле"),
76) positive("Наблюдается","слюнотечение"),
77) not(positive("Имеется","высокая температура")),
78) asserta(found("Инородное тело в гортани")),!.
79) test4("n","n","n","n").
80) test5("a","n","n","n","m") :-
81) positive("Чувствуются","боли в ухе"),
82) positive("Наблюдаются","симптомы интоксикации"),
83) asserta(found("Отит")),!.
84) test5("n","n","n","n","l") :-
85) positive("Чувствуется","тошнота"),
86) positive("Чувствуется",
87) "понижение слуха при попадании в ухо воды"),
88) positive("Чувствуется","головная боль"),
89) asserta(found("Серная пробка в ухе")),!.
90) it_is("Предположение1") :-
91) positive("Имеется","высокая температура"),
92) positive("Чувствуется","головная боль").
93) it_is("Предположение2") :-
94) positive("Наблюдается","насморк"),
95) positive("Наблюдается","кашель").
96) GOAL
97) run.
```

Сравнение прямой и обратной стратегий логического вывода

Количество вопросов, которые задаются пользователю, в среднем оказывается меньшим для прямой стратегии, так как в ней реализован механизм, похожий на двоичный поиск, на каждом шаге которого отбрасывается примерно половина возможных решений. Однако реализация прямой стратегии требует более тщательного анализа признаков и их

логических сочетаний с тем, чтобы составить блок-схему алгоритма такого механизма поиска. Кроме того, содержание и последовательность вопросов могут показаться пользователю нелогичными, так как эти вопросы не связаны напрямую с возможным решением. В обратной стратегии содержание и последовательность вопросов сразу ориентированы на текущее предположение, которое пытается проверить программа. Поэтому эти вопросы воспринимаются пользователем более естественно, так как у него есть возможность догадаться, какое предположение система пытается проверить в текущий момент. Но количество этих вопросов в общем случае может оказаться большим, чем в случае использования прямой стратегии.

Исходя из принципов организации каждой из стратегий, можно предложить использование **прямой стратегии для задач, в которых количество исходных данных обозримо, а количество возможных решений достаточно велико**.

Обратную стратегию целесообразно применять в обратном случае, т. е. **когда количество решений невелико, а количество требующих проверки признаков немало и их логические взаимосвязи довольно сложны.**

При решении задач, к которым можно применить обе стратегии, предпочтение следует отдавать обратной стратегии вывода по следующим причинам:

- При реализации обратной стратегии отсутствует этап составления блок-схемы последовательности проверок. Все имеющиеся признаки достаточно привязать к тому или иному решению без построения какой-либо неестественной иерархии.
- Исходный код программы для обратной стратегии легко модифицируется в случае необходимости добавления признаков, не учтенных ранее. В случае прямой стратегии для этого требуется переделка схемы алгоритма проверок, что может привести к возврату на этап ее составления.
- При программной реализации обратной стратегии на языке Пролог она получается более адекватной, так как в этом случае совпадают механизмы вывода в программе и в Пролог-системе, а это приводит к уменьшению трудоемкости дальнейшего развития программы.